CBOR

Concise Binary Object Representation (**CBOR**) is a binary data serialization format loosely based on JSON authored by C. Bormann. Like JSON it allows the transmission of data objects that contain name–value pairs, but in a more concise manner. This increases processing and transfer speeds at the cost of human readability. It is defined in IETF RFC 8949 (https://datatracker.ietf.org/doc/html/rfc8949).[1]

CBOR	
Filename extension	`.cbor`
Internet media type	`application/cbor`
Type of format	Data interchange
Extended from	MessagePack
Standard	RFC 8949 (https://datatracker.ietf.org/doc/html/rfc8949)
Open format?	Yes
Website	cbor.io (https://cbor.io/)

Amongst other uses, it is the recommended data serialization layer for the CoAP Internet of Things protocol suite[2] and the data format on which COSE messages are based. It is also used in the Client-to-Authenticator Protocol (CTAP) within the scope of the FIDO2 project.[3]

CBOR was inspired by MessagePack, which was developed and promoted by Sadayuki Furuhashi. CBOR extended MessagePack, particularly by allowing to distinguish text strings from byte strings, which was implemented in 2013 in MessagePack.[4][5]

Specification of the CBOR encoding

CBOR encoded data is seen as a stream of data items. Each data item consists of a header byte containing a 3-bit type and 5-bit short count. This is followed by an optional extended count (if the short count is in the range 24–27), and an optional payload.

For types 0, 1, and 7, there is no payload; the count *is* the value. For types 2 (byte string) and 3 (text string), the count is the length of the payload. For types 4 (array) and 5 (map), the count is the number of items (pairs) in the payload. For type 6 (tag), the payload is a single item and the count is a numeric tag number which describes the enclosed item.

CBOR data	**Data item 1**				**Data item 2**				**Data item 3...**
Byte count	1 byte (CBOR data item header)		Variable	Variable	1 byte (CBOR data item header)		Variable	Variable	etc...
Structure	Major type	Short count	Extended count (optional)	Data payload (optional)	Major type	Short count	Extended count (optional)	Data payload (optional)	etc...
Bit count	3 Bits	5 Bits	8 Bits × variable	8 Bits × variable	3 Bits	5 Bits	8 Bits × variable	8 Bits × variable	etc..

Major type and count handling in each data item

Each data item's behaviour is defined by the major type and count. The major type is used for selecting the main behaviour or type of each data item.

The 5-bit short count field encodes counts 0–23 directly. Short counts of 24–27 indicate the count value is in a following 8, 16, 32 or 64-bit extended count field. Values 28–30 are not assigned and must not be used.

Types are divided into "atomic" types 0–1 and 6–7, for which the count field encodes the value directly, and non-atomic types 2–5, for which the count field encodes the size of the following payload field.

A short count of 31 is used with non-atomic types 2–5 to indicate an indefinite length; the payload is the following items until a "break" marker byte of 255 (type=7, short count=31). A short count of 31 is not permitted with the other atomic types 0, 1 or 6.

Type 6 (tag) is unusual in that its count field encodes a value directly, but also has a payload field (which always consists of a single item).

Extended counts, and all multi-byte values, are encoded in network (big-endian) byte order.

CBOR data item field encoding

Tiny Field Encoding

Byte count	1 byte (CBOR data item header)	
Structure	Major type	Short count (Value)
Bit count	3 Bits	5 Bits
Atom	0–1, 7	0–23
Break marker	7	31

Short Field Encoding

Byte count	1 byte (CBOR data item header)		Variable
Structure	Major type	Short count	Value
Bit count	3 Bits	5 Bits	8 Bits × variable
Atom	0–1, 7	24–27	8, 16, 32 or 64 bits
String	2–3	0–23	count × 8 bits
Items	4–5	0–23	count × items/pairs
Tag	6	0–23	one item

Long Field Encoding

Byte count	1 byte (CBOR data item header)		1, 2, 4 or 8 bytes	Variable
Structure	Major type	Short count (24–27)	Extended count (Length of payload)	Value
Bit count	3 Bits	5 Bits	8, 16, 32 or 64 bits	8 Bits × vari
String	2–3	24–27	Up to 2^{64}–1	count × 8 bits
Items	4–5	24–27	Up to 2^{64}–1	count × items/pairs
Tag	6	24–27	Tag, up to 2^{64}–1	one item

Integers (types 0 and 1)

For integers, the count field *is* the value; there is no payload. Type 0 encodes positive or unsigned integers, with values up to 2^{64}–1. Type 1 encodes negative integers, with a value of −1–count, for values from -2^{64} to −1.

Strings (types 2 and 3)

Types 2 and 3 have a count field which encodes the length in bytes of the payload. Type 2 is an unstructured byte string. Type 3 is a UTF-8 text string.

A short count of 31 indicates an indefinite-length string. This is followed by zero or more definite-length strings of the same type, terminated by a "break" marker byte. The value of the item is the concatenation of the values of the enclosed items. Items of a different type, or nested indefinite-length strings, are not permitted. Text strings must be individually well-formed; UTF-8 characters may not be split across items.

Arrays and maps (types 4 and 5)

Type 4 has a count field encoding the number of following items, followed by that many items. The items need not all be the same type; some programming languages call this a "tuple" rather than an "array".

Alternatively, an indefinite-length encoding with a short count of 31 may be used. This continues until a "break" marker byte of 255. Because nested items may also use the indefinite encoding, the parser must pair the break markers with the corresponding indefinite-length header bytes.

Type 5 is similar but encodes a map (also called a dictionary, or associative array) of key/value pairs. In this case, the count encodes the number of *pairs* of items. If the indefinite-length encoding is used, there must be an even number of items before the "break" marker byte.

Semantic tag (type 6)

A semantic tag is another atomic type for which the count is the value, but it also has a payload (a single following item), and the two are considered one item in e.g. an array or a map.

The tag number provides additional type information for the following item, beyond what the 3-bit major type can provide. For example, a tag of 1 indicates that the following number is a Unix time value. A tag of 2 indicates that the following byte string encodes an unsigned bignum. A tag of 32 indicates that the following text string is a URI as defined in RFC 3986 (https://datatracker.ietf.org/doc/html/rfc3986). RFC 8746 (https://datatracker.ietf.org/doc/html/rfc8746) defines tags 64–87 to encode homogeneous arrays of fixed-size integer or floating-point values as byte strings.

The tag 55799 is allocated to mean "CBOR data follows". This is a semantic no-op, but allows the corresponding tag bytes `d9 d9 f7` to be prepended to a CBOR file without affecting its meaning. These bytes may be used as a "magic number" to distinguish the beginning of CBOR data.

The all-ones tag values 0xffff, 0xffffffff and 0xffffffffffffffff are reserved to indicate the absence of a tag in a CBOR decoding library; they should never appear in a data stream.

The break marker pseudo-item may not be the payload of a tag.

Special/float (type 7)

This major type is used to encode various special values that do not fit into the other categories. It follows the same encoding-size rules as the other atomic types (0, 1, and 6), but the count field is interpreted differently.

The values 20–23 are used to encode the special values false, true, null, and undefined. Values 0–19 are not currently defined.

A short count of 24 indicates a 1-byte extended count follows which can be used in future to encode additional special values. To simplify decoding, the values 0–31 may not be encoded in this form. None of the values 32–255 are currently defined.

Short counts of 25, 26 or 27 indicate a following extended count field is to be interpreted as a (big-endian) 16-, 32- or 64-bit IEEE floating point value. These are the same sizes as an extended count, but are interpreted differently. In particular, for all other major types, a 2-byte extended count of 0x1234 and a 4-byte extended count of 0x00001234 are exactly equivalent. This is not the case for floating-point values.

Short counts 28–30 are reserved, like for all other major types.

A short count of 31 encodes the special "break" marker which terminates an indefinite-length encoding. This is related to, but different from, the use with other major types where a short count of 31 *begins* an indefinite length encoding. This is not an item, and may not appear in a defined-length payload.

Semantic tag registration

IANA has created the CBOR tags registry, located in https://www.iana.org/assignments/cbor-tags/cbor-tags.xhtml . Registration must contain these template.[6]

Semantic tag type	Range	Template			
		Data item	Semantic description (Short Form)	Point of contact	Description of semantics (URL)
Standard actions	0–23	Required	Required	N/A	N/A
Specification required	24–255	Required	Required	N/A	N/A
First Come First Served	256–18446744073709551615	Required	Required	Required	Description is optional. The URL can point to an Internet-Draft or a web page.

Implementations

Name	Primary author	Language	License	Source	Remarks
cbor	Kyunghwan Kwon	C	MIT	https://github.com/mononn/cbor	
QCBOR	Laurence Lundblade	C	MIT	https://github.com/laurencelundblade/QCBOR	
cbor-js	Patrick Gansterer	JavaScript	MIT	https://github.com/paroga/cbor-js	
node-cbor	Joe Hildebrand	JavaScript	MIT	https://github.com/hildjj/node-cbor	
CBOREncode	Pavel Gulbin	PHP	PHP	https://github.com/2tvenom/CBOREncode	
cbor-php	Florent Morselli	PHP	MIT	https://github.com/Spomky-Labs/cbor-php	
fxamacker/cbor	Faye Amacker	Go	MIT	https://github.com/fxamacker/cbor	Fuzz tested, RFC 8949, CBOR tags, Core Deterministic Encoding, float64/32/16, duplicate map key detection, API is encoding/json + toarray & keyasint struct tags, etc.
cbor	Pavel Gulbin	Go	WTFPL	https://github.com/2tvenom/cbor	
cbor_go	Brian Olson	Go	APL 2.0	https://github.com/brianolson/cbor_go	
go-codec	Ugorji Nwoke	Go	MIT	https://godoc.org/github.com/ugorji/go/codec	Also handles JSON, MsgPack and BinC.
serde_cbor	Pyfisch	Rust	MIT or APL 2.0	https://github.com/pyfisch/cbor	
cbor-codec	Toralf Wittner	Rust	MPL 2.0	https://twittner.gitlab.io/cbor-codec/cbor/	
SwiftCBOR	greg@ unrelenting.technology	Swift	Unlicense	https://github.com/unrelentingtech/SwiftCBOR	
CBOR.jl	Saurav Sachidanand	Julia	MIT	https://github.com/saurvs/CBOR.jl	
Lua-CBOR	Kim Alvefur	Lua	MIT	https://www.zash.se/lua-cbor.html	
org.conman.cbor	Sean Conner	Lua	GNU LGPL-3	https://github.com/spc476/CBOR	
cbor_py	Brian Olson	Python	APL 2.0	https://github.com/brianolson/cbor_py	
flynn	Fritz Conrad Grimpen	Python	MIT	https://github.com/fritz0705/flynn	
cbor2	Alex Grönholm	Python	MIT	https://github.com/agronholm/cbor2	

Name	Primary author	Language	License	Source	Remarks
CBOR::Free	Felipe Gasper	Perl	GNU GPL & Artistic	https://metacpan.org/pod/CBOR::Free	
CBOR::PP	Felipe Gasper	Perl	GNU GPL & Artistic	https://metacpan.org/pod/CBOR::PP	
CBOR::XS	Marc Lehmann	Perl	GNU GPL-3	https://metacpan.org/pod/CBOR::XS	
cbor-ruby	Sadayuki Furuhashi Carsten Bormann	Ruby	APL 2.0	https://github.com/cabo/cbor-ruby	
libcbor-ruby	Pavel Kalvoda	Ruby	MIT	https://github.com/PJK/libcbor-ruby	Binding to libcbor.
cbor-erlang	Jihyun Yu	Erlang	BSD-3-clause	https://github.com/yjh0502/cbor-erlang	
excbor	Carsten Bormann	Elixir	not specified, ask the author	https://github.com/cabo/excbor	
CBOR	R. Kyle Murphy	Haskell	GNU LGPL-3	https://github.com/orclev/CBOR	
borc	Joe Hildebrand Friedel Ziegelmayer	JavaScript	MIT	https://github.com/dignifiedquire/borc	Fork of node-cbor.
borc-refs	Joe Hildebrand Friedel Ziegelmayer Sandro Hawke	JavaScript	MIT	https://github.com/sandhawke/borc-refs	Fork of borc.
CBOR	Peter Occil	C#	Public domain software	https://github.com/peteroupc/CBOR	Also handles JSON.
Dahomey.Cbor	Michaël Catanzariti	C#	MIT License	https://github.com/dahomey-technologies/Dahomey.Cbor	
Jackson	Tatu Saloranta	Java	APL-2.0	https://github.com/FasterXML/jackson-dataformats-binary/tree/master/cbor	Also handles other formats.
cbor-java	Constantin Rack	Java	APL-2.0	https://github.com/c-rack/cbor-java	
jacob	J.W. Janssen	Java	APL-2.0	https://github.com/jawi/jacob	
kotlinx.serialization	JetBrains	Kotlin	APL-2.0	https://github.com/Kotlin/kotlinx.serialization	Supports cross-platform

Name	Primary author	Language	License	Source	Remarks
cn-cbor	Joe Hildebrand Carsten Bormann	C	MIT	https://github.com/cabo/cn-cbor	
cbor-cpp	Stanislav Ovsyannikov	C++	APL-2.0	https://github.com/naphaso/cbor-cpp	
cppbor	David Preece	C++	BSD	https://github.com/rantydave/cppbor	Uses C++17 variants.
libcbor	Pavel Kalvoda	C	MIT	https://github.com/PJK/libcbor	
tinycbor	Intel	C	MIT	https://github.com/01org/tinycbor	
NanoCBOR	Koen Zandberg	C	Public domain	https://github.com/bergzand/NanoCBOR	Used by RIOT-OS
cbor-d	Andrey Penechko	D	Boost 1.0	https://github.com/MrSmith33/cbor-d	
clj-cbor	Greg Look	Clojure	Unlicense	https://github.com/greglook/clj-cbor	
JSON for Modern C++	Niels Lohmann	C++	MIT	https://github.com/nlohmann/json	Also handles JSON and MsgPack.
borabora	Christoph Engelbert	Java	APL-2.0	https://github.com/noctarius/borabora	
lua-ConciseSerialization	François Perrad	Lua	MIT	https://fperrad.frama.io/lua-ConciseSerialization/	
flunn	Fritz Conrad Grimpen Sokolov Yura	Python	MIT	https://pypi.python.org/pypi/flunn	
cbor-qt	Anton Dutov	C++	Public domain	https://github.com/anton-dutov/cbor-qt	
QCborValue	Qt Project	C++	GNU LGPL	https://doc.qt.io/qt-5/qcborvalue.html	Part of the Qt framework since version 5.12
cbor11	Jakob Varmose Bentzen	C++	Public domain	https://github.com/jakobvarmose/cbor11	
cborcpp	Alex Nekipelov	C++	MIT	https://github.com/nekipelov/cborcpp	
GoldFish	Vincent Lascaux	C++	MIT	https://github.com/OneNoteDev/GoldFish	
Library-Arduino-Cbor	Juanjo Tara	C++	APL-2.0	https://github.com/jjtara/Library-Arduino-Cbor	
cborg	Duncan Coutts	Haskell	BSD-3-clause	https://github.com/well-typed/cborg	
cbor	Steve Hamblett	Dart	MIT	https://github.com/shamblett/cbor	
borer	Mathias Doenitz	Scala	MPL 2.0	https://github.com/sirthias/borer	Also handles JSON.

Name	Primary author	Language	License	Source	Remarks
nim_cbor	Emery Hemingway	Nim	MIT	https://git.sr.ht/~ehmry/nim_cbor	
ciborium	Nathaniel McCallum Mark Bestavros Enarx Team	Rust	Apache 2.0	https://github.com/enarx/ciborium	
cbor	Paulo Moura	Logtalk	Apache 2.0	https://github.com/LogtalkDotOrg/logtalk3/tree/master/library/cbor	Part of the Logtalk distribution; can also be used from Prolog
System.Formats.Cbor	.NET Team	C#	MIT	https://github.com/dotnet/runtime/blob/main/src/libraries/System.Formats.Cbor	Part of .NET 5+
DelphiCBOR	mikerabat	Delphi	Apache License 2.0	https://github.com/mikerabat/DelphiCBOR	

See also

- Comparison of binary data serialization formats

References

1. "CBOR — Concise Binary Object Representation | Overview" (http://cbor.io/).
2. "CoAP — Constrained Application Protocol | Overview" (http://coap.technology/).
3. "FIDO2 Project" (https://fidoalliance.org/fido2/). FIDO Alliance. Retrieved 2018-05-11.
4. "Discussions on the upcoming MessagePack spec that adds the string type to the protocol" (https://github.com/msgpack/msgpack/issues/128). *GitHub*. Retrieved 2022-01-04.
5. "RFC 8949: Concise Binary Object Representation (CBOR)" (https://www.rfc-editor.org/rfc/rfc8949.html). IETF. December 2020.
6. https://www.rfc-editor.org/rfc/rfc8949.html#name-cbor-tags-registry

External links

- Online tool to convert from CBOR binary to textual representation and back. (http://cbor.me/)